NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

PERFORMANCE ASSESSMENT OF NETWORK INTRUSION-ALERT PREDICTION

by

Farn Wei Jason Khong

September 2012

Thesis Advisor:	Christian J. Darken
Thesis Co-Advisor:	Neil C. Rowe
Second Reader:	Terence Tan

This thesis was performed at the MOVES Institute
Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2012	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE Performance Assessment of Network Intrusion-Alert Prediction			**5. FUNDING NUMBERS**
6. AUTHOR(S) Farn Wei Jason Khong			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ___N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

In the current global cyber warfare landscape, cyber attacks on infrastructure are a serious threat. Although network administrators use intrusion detection systems (IDSs) to detect threats and anomalies, they usually only offer post-attacks alerts. If we could predict malicious activities, we could allow network administrators or security enhancing software to take appropriate actions in advance of damage occurring. Incoming intrusion detection alerts can be considered as a sequence. We used Pytbull to simulate cyber attacks within a testbed network environment and collected Snort generated intrusion detection alerts. We tested four sets of alert-prediction programs with this data: Single-Scope Blending algorithm, a Simple Bayesian Mixture algorithm, a Multiple Simple Bayesian algorithm and a Variable Markov Model algorithm. The harmonic mean of the precision and recall (F-score) measured prediction accuracy. The Single-Scope Blending algorithm performed the best in these tests, especially in a multiple attacker environment.

14. SUBJECT TERMS Artificial Intelligence, Agent-Based Modeling/Simulation, Network Security, Intrusion Detection System, Alert Prediction			**15. NUMBER OF PAGES** 59
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

PERFORMANCE ASSESSMENT OF NETWORK INTRUSION-ALERT PREDICTION

Farn Wei Jason Khong
Civilian, Defence Science and Technology Agency, Singapore
B.Eng., (Hons), Nanyang Technological University, 2007

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN
MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
September 2012

Author: Farn Wei Jason Khong

Approved by: Christian J. Darken
Thesis Advisor

Neil C. Rowe
Thesis Co-Advisor

Kian-Moh Terence Tan
Second Reader

Christian J. Darken
Chair, MOVES Academic Committee

Peter J. Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In the current global cyber warfare landscape, cyber attacks on infrastructure are a serious threat. Although network administrators use intrusion detection systems (IDSs) to detect threats and anomalies, they usually only offer post-attacks alerts. If we could predict malicious activities, we could allow network administrators or security enhancing software to take appropriate actions in advance of damage occurring. Incoming intrusion detection alerts can be considered as a sequence. We used Pytbull to simulate cyber attacks within a testbed network environment and collected Snort generated intrusion detection alerts. We tested four sets of alert-prediction programs with this data: Single-Scope Blending algorithm, a Simple Bayesian Mixture algorithm, a Multiple Simple Bayesian algorithm and a Variable Markov Model algorithm. The harmonic mean of the precision and recall (F-score) measured prediction accuracy. The Single-Scope Blending algorithm performed the best in these tests, especially in a multiple attacker environment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

Acronym	Definition
CSV	Comma-separated Values
IP	Internet Protocol
MSB	Multiple Simple Bayesian
RTS	Relational Time Series
SBM	Simple Bayesian Mixture
SQL	Structured Query Language
SSB	Single-Scope Blending
VOMM	Variable Order Markov Models

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This master's thesis would not have been possible without the help and support of the people of numerous people. I would like to express my gratitude to my thesis advisor, Dr. Christian Darken, for his guidance and encouragement. His lectures and work on Artificial Intelligence spurred me to work on this thesis. I would also like to thank my thesis co-advisor, Professor Neil Rowe, especially for sharing his technical expertise and guidance on cyber security and intrusion-detection systems. Many thanks to my second reader, Mr Terence Tan, for providing his prediction algorithm codes to make the evaluation possible and also in his relentless help in explaining hours the theories.

I would also like to thank my sponsor, the Defence Science and Technology Agency (DSTA) for making my studies at the Naval Postgraduate School possible.

I wish to thank my friends and colleagues for their encouragements and well-wishes.

Finally, I would like to thank my family and my wife, who are in Singapore, for their love, patience and being understanding during my stint in Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In his statement that “cyber threat is one of the most serious economic and national security challenges we face as a nation,” President Obama stressed the importance of securing networks against cyber attacks (Council, n.d.). Network administrators employ intrusion-detection systems (IDSs) to detect malicious threats on their computer networks. The intrusion-detection system monitors and generates alerts for network traffic that are malicious or suspicious (Albin, 2011). However, if an attack is genuine, the system usually reports the attack only after it has happened since most attacks happen in seconds and most intrusion-detection systems are not linked to an intrusion prevention system that takes immediate action. Network administrators will then take steps to rectify any system malfunction caused by the event by inspecting the alerts later. It would reduce or even prevent damages if one could predict the attack and perform pre-emptive actions.

The goal of the thesis was to provide data to compare the performance of several prediction algorithms that could infer alerts earlier. These algorithms have various degrees of success in predicting states, events and actions on an agent-based simulation system (Tan & Darken, 2012a, 2012b). Tan adapted the programs used in Tan and Darken (2012a, 2012b) to predict Snort alerts. We evaluate the prediction algorithms by running the programs provided by Tan to compare their prediction accuracy on the effects of different attack configurations. The research required a collection a representative set of intrusion-detection system alert logs as the dataset for processing by the prediction algorithms. Computer networks are constantly exposed to cyber attacks. This threat has been growing over the years in terms of attack frequency and damage level. According to Symantec (2011), there are more than 286 million new threats in 2010. The U.S. Computer Emergency Readiness Team (US-CERT) reported a 40 percent increase in cyber attacks in 2010 on federal agencies, from 30,000 the previous year to 41,766 (Johnson, 2011).

Traditionally, damages arising from cyber attacks range from taking Internet services offline to classified company information leaks, loss of personnel information, credit card information theft, etc. The focus has shifted to attacks that can cause significant damage.

Intrusion detection alerts can be expressed as a Relational Time Series (RTS). The intrusion-detection system generates alerts as malicious activities arrive in time sequence. According to Tan and Darken (2012a, 2012b), a RTS is a “sequence of relational percepts.” (Tan & Darken, 2012b). The intrusion detection alerts RTS is inherently unknown, noisy and constantly evolving. Hence an alerts RTS provides a good domain for evaluating the effectiveness of new prediction algorithms.

Chapter II describes the background of prediction algorithms. We will discuss the background of the key components used for the thesis in Chapter III. Chapter IV presents the steps involved in the setup of the experiment, details of the software, hardware and individual component configurations. Chapter V provides the evaluation of the results and Chapter VI provides the conclusion and suggestions for future work.

II. BACKGROUND ON ALERT PREDICTION

Intrusion-detection systems generate alerts when attack activities have taken place. They allow network administrators to conduct remedial actions. However, intrusion-detection systems cannot predict attack activities. A proactive approach is to anticipate and conduct possible attacks to prevent damage. This chapter describes the current approaches to prediction algorithm that may be applicable to predicting intrusion detection alerts.

One approach to predicting an attacker's behavior is plan recognition. Geib and Goldman (2001) defined a plan library of specific attacks to predict an attack plan. Plan recognition entails having a security professional to compile the plan library manually. It is time consuming and not always able to respond to new attack variants. To account for variation in order or missing actions in an attack sequence, will increase the complexity of the plan matching. Also, the plan library must be updated frequently to meet new attack sequence.

Other methods do data mining to predict the occurrence likelihood of the next alert. Cipriano, Zand, Houmansadr, Kruegel, and Vigna (2011) introduced such a prediction algorithm, Nexat, that automates machine learning process. During data mining, it uses historical data to learn the co-occurrence of the alerts. At run time, it uses the trained database and weighted probability to predict the next alert. A large database of historical data is required. Nexat finds a fit to the historical data and so cannot predict new attacks.

Other work proposed proposed the use of "network attack graph" to analyze the security vulnerabilities and find all possible attack sequences (Lei & Li, 2007). A network attack graph is generated by correlating alerts according to source and destination Internet Protocol (IP) addresses. The predicted next alert is determined through predictability scores derived from the attack graph. It provides graphical flow of the attack sequence to the network administrator. However, the graph generation process includes low probable alerts into the

attack sequence, which must then be removed manually to improve the prediction. This method also cannot detect out of sequence attacks.

Another technique called “sequence pattern mining” reduces the effort to construct pattern rules. Using the database derived from a historical attack sequence is vulnerable to new attack strategies. Li, Zhang, Li, and Wang (2007) observe that most attacks are completed within a certain time span. They proposed an incremental mining algorithm to identify sequential attack patterns over divided time window. The database is updated within a shorter period after the new attack strategy appears. After the initial rule generation, the performance of subsequent updates would be faster as the number of new alert sequence received reduces.

Another way of processing security alerts is by organizing them into relational time series (RTS). The intrusion-detection system generates security alerts in a sequential order by time of arrival. These alerts form a sequence of relational percepts. “Each percept is a ground atom defined as $p_i = r(c_1, c_2, \ldots, c_m)$, where r is the predicate and $c_{j\in(1..m)}$ are constants that represent objects” (Tan & Darken, 2012b). For security alerts, r is the alert type/identity and $c_{j\in(1..m)}$ refers to an entity such as source or destination IP. We give an example of this representation for a stream of alerts in Table 1.

Time	Incoming Security Alerts	Relational Time Series
0	Alert message: (spp_frag3) Short fragment, possible DoS attempt Source: 192.168.1.2 Destination: 192.168.1.3	ShortFragDOS(192.168.1.2, 192.168.1.3, UDP)
21	Alert message: Reset outside window Source: 192.168.1.2 Destination: 192.168.1.3	ResetWindow(192.168.1.2, 192.168.1.3, TCP)
30	Alert message: ICMP-INFO Fragment Reassembly Time Exceeded Source: 192.168.1.3 Destination: 192.168.1.2	FragReassemblyExceed(192.168.1.3, 192.168.1.2, ICMP)
38	Alert message: Reset outside window Source: 192.168.1.2 Destination: 192.168.1.3	ResetWindow(192.168.1.2, 192.168.1.3, TCP)
38.5	Alert message: (spp_frag3) Fragmentation overlap Source: 192.168.1.2 Destination: 192.168.1.3	FragOverlap(192.168.1.2, 192.168.1.3, UDP)

Table 1. Example of Network Security Alerts in Relational Time Series

A software agent can learn percepts based on the situation (situation learning) and can predict future events in a RTS (Darken, 2005). When predicting the next percept, we can take into account all previous percept sequences to derive a probability distribution for a prediction. A simplifying assumption is that recent percepts are more useful than all the percepts. This is relevant to cyber-attack activities where related attack events generally arrive within a short time span (“situation-based learning”). This also helps with noisy network traffic by reducing stray alerts from the predictor function. The situation learning approach organizes the RTS into smaller grouped situations. In addition to increased predictor relevancy (in terms of recent percepts), situation learning reduces the prediction complexity. Situation learning can be accomplished by a variety of

inferencing methods such as Variable Order Markov Models (VOMM), Multiple Simple Bayesian (MSB), Simple Bayesian Mixture (SBM) and Single-Scope Blending (SSB).

Tan and Darken (2012a) compared the prediction performance of these methods in a role-playing game environment, where an agent moves and perform actions randomly together with other agents. In Multiple Siple Bayesian inference, there is a naïve Bayesian network for each predictive percept and situation pair. During each prediction event, the Bayesian network forms a probability distribution for all previously seen alerts by computing $P(A_i|C)$, where P is the conditional probability, A_i refers to each alerts observed, C is the current situation. Simple Bayesian Mixture inference is implemented by normalizing a linear combination of multiple probability densities. Variable Order Markov Models use a variable order Markov chain instead of a fixed order.

Single-Scope Blending inference is shown in Figure 1. A “generic space” contains the common atoms in both concept 1 and concept 2. Concept 2 is the current situation, and concept 1 is a previous situation that is selected to maximize the generic space. That is, concept 1 is the most similar situation. Blend B is the predicted situation which is generated by using the frame from concept 1 and constant mapping from concept 2. This is a form of inference by analogy.

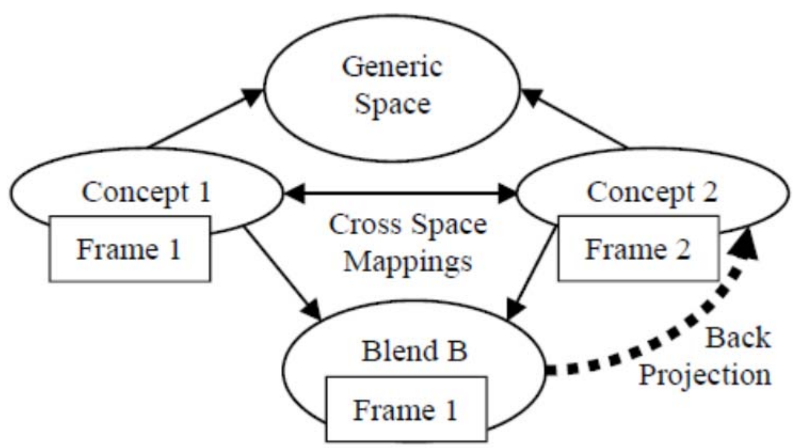

Figure 1. Single-Scope Blending Network (From Tan & Darken, 2012b)

Single-Scope Blending could achieve better prediction performance than the other inference methods because it makes use of similarities for prediction instead of exact matching (Tan & Darken, 2012b). Since intrusion detection alerts form a RTS sequence, Single-Scope Blending would seem promising. Thus Tan adapted the programs used in Tan and Darken (2012a, 2012b) to predict Snort alerts. We tested these programs to evaluate their performance. A collection of intrusion-detection datasets was required. We generated these alerts within a controlled environment using the tools described in Chapter III.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

A. HONEYPOTS

A collection of datasets (intrusion detection alerts) is required for the prediction-algorithms analysis. One way to gather the dataset is through honeypots, machines explicitly designated solely to learn the methods used by black-hats to probe and hack a system so that a network administrator can improve the security policies (Spitzner, 1999). Monitoring tools, such as an intrusion-detection system, are installed on the honeypot. If placed within the network, honeypots are used to monitor abnormal activities such as compromised systems within the organization. Our first experiments used such data.

However, with honeypots we cannot control important factors that may affect prediction performance, such as frequency of attacks, number of attackers and number of targets. There is also noise traffic in honeypot data which makes analysis difficult. Collection of data sufficient for analysis through deployment of honeypots can be time consuming. Also, as vulnerabilities of the honeypot are learned, hackers may give up and go after easier targets, which decreases the alerts logged (Rowe, Custy, & Duong, 2007).

Therefore, the thesis explored an alternative of simulating honeypot data. We controlled the environment to provide data on only specific types of attacks. This minimized “noise traffic” as the intrusion-detection system was not directly exposed to the Internet. The testbed environment consisted of a local-area network, the intrusion-detection system and the attackers on other machine. The attacks were carried out in various configurations and we kept a log file of the alerts produced. These log files are used as the dataset for the prediction algorithms.

B. INTRUSION DETECTION SYSTEM

1. Intrusion-Detection Techniques

Intrusion-detection techniques are anomaly-based and/or signature-based. Anomaly-based detection examines the operation profile of the network and determines what considers the normal activities. A deviation from the operation profile causes the intrusion-detection system to send an alarm for anomaly activities. Signature-based detection, also known as rule-based detection, uses information of historical malicious activities as signatures to determine the threats. In this thesis, we use Snort to generate the intrusion detection alerts.

2. SNORT

> Snort is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire. Combining the benefits of signature, protocol, and anomaly-based inspection, Snort is the most widely deployed IDS/IPS technology worldwide. With millions of downloads and nearly 400,000 registered users, Snort has become the de facto standard for IPS. (Snort, 2012)

We choose Snort because it is an open source product that is free to download and can be deployed cross-platform (Windows and Linux). It can be installed and run from a personal computer. The Sourcefire Vulnerability Research Team provides tested and certified rules free for registered users. The rules are updated regularly. A subscription is required for latest initial release. The rules are available to registered users after 30 days of initial release. Snort monitors the network and detects known threats using signatures and threat patterns.

We briefly describe the Snort architecture (Figure 2) Snort consists of four main components (Olney, 2008):

- Packet decoder. The key requirement of Snort is to capture network packets. Libpcap (for Linux) or Winpcap (for Windows) must be installed for packet capturing. The packet decoder translates it into packet-header information and payload.

- Preprocessors. The preprocessors rearrange or reassemble packets before the detection engine analyzes them. Incoming packets may be fragmented to avoid detection by the standard Snort rules, so preprocessors reassemble fragmented packets and generate pseudo packets to be fed back to the packet decoder.
- Detection engine. The detection engine analyzes all packets with pre-defined rules. If a match is found, the packet is sent to the output module. The rule syntax can include various elements in a data packet such as protocol type, port number, packet length, packet header and packet content.
- Outputs. After a threat is detected, the information is passed to the output module for presentation. An alert can be sent to the administrators by pop-up messages or email alerts. The alerts can be stored on a text file, csv (comma-separated values) file or on a Structured Query Language (SQL) database. Our research stored the generated alerts into csv files.

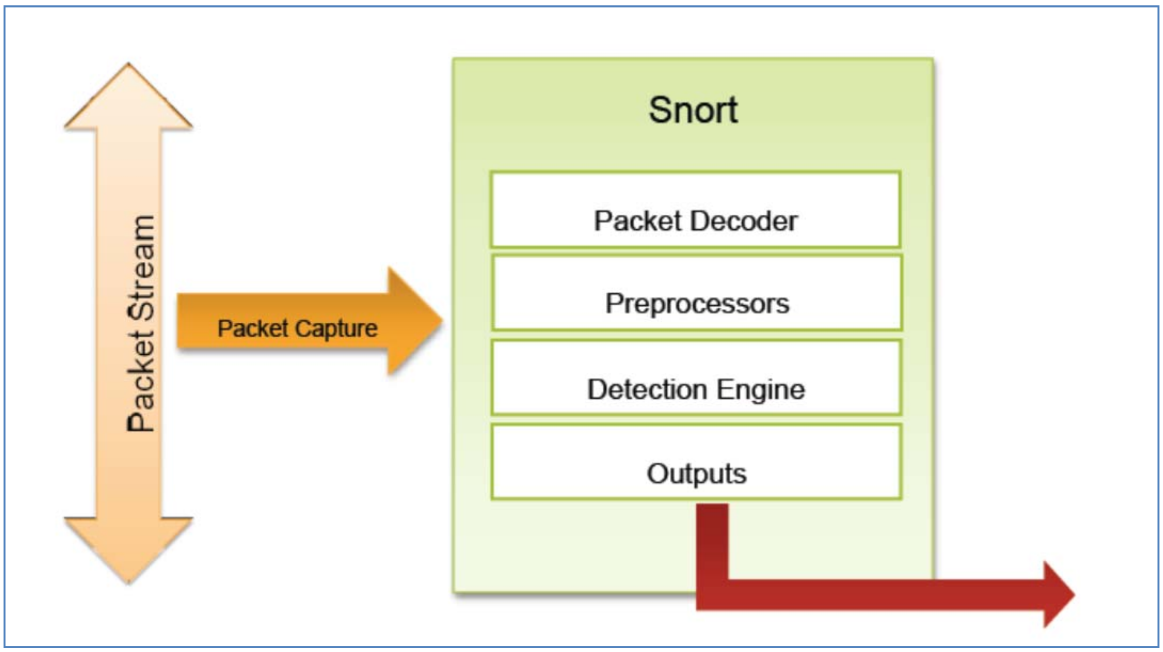

Figure 2. Snort Architecture (From Olney, 2008)

We deployed Snort by connecting it to a port-mirroring switch. We configured the switch to send a copy of every network packet of other ports to the mirrored port. The test environment entails both attacker and target machines within the local-area network.

C. PYTBULL

To create various attack configurations for prediction performance analysis, we use tools to simulate malicious traffic in an experimental network. A intrusion-prevention system penetration tester can do this. It injects malicious packets into the network either by means of custom packets with attack signatures or simulating attack patterns. We used Pytbull to do this and yield an alert file (Damaye, 2012).

Pytbull can automatically conduct simulated attacks on a target. A Pytbull application consists of an attacker machine and a server. The prerequisite services running on the server are FTP, HTTP, SSH and the Pytbull server itself (running a reverse shell). These services allow Pytbull, executing from the attacker machine, to conduct tests related to these services. Pytbull provides about 300 tests in 11 testing modules, listed in Table 2. These modules are reconfigurable, which allow us to customize the attack patterns.

No.	**Test Module**	**Description**
1	badTraffic	Non-RFC-compliant packets are sent to the server.
2	bruteForce	Tests the ability of the server to track brute force attacks (as on FTP).
3	clientSideAttacks	Uses a reverse shell to provide the server with instructions to download remote malicious files.
4	denialOfservice	Tests the ability of the intrusion-detection system to protect against denial-of-service attempts.
5	evasionTechniques	Check if the intrusion-detection system can detect various evasion methods.
6	fragmentedPackets	Sends various fragmented payloads to the server to test its ability to recompose them and detect attacks.
7	ipReputation	Tests the ability of the server to detect traffic from/to low reputation servers.
8	normalUsage	Sends payloads that correspond to normal usage.
9	pcapReplay	Repalys pcap files (packet sequences)
10	shellCodes	Sends various shellcodes to the server on port 21/tcp to test its ability to reject them.
11	testRules	Testing of basic rules. of the intrusion-detection system/intrusion prevention system.

Table 2. Pytbull Test Modules (From Damaye, 2012)

Each testing module allows tests that to be enabled or disabled via configuration files. Pytbull conducts these tests in sequence. At the end of each run, we extracted the alert log file to determine whether the tests are detected.

Similar experiments were conducted in (Albin, 2011), which identified tests such as client-side attacks and pcap replay (pcap of the Slammer worm) that were not detected by Snort, although a large number of repetitive and not meaningful "reset outside window" alerts were found in our experiment. We excluded these tests from our random attacks run to reduce the number of "reset outside window" alerts. We broke down the individual attacks into separate configuration files so that we could select or randomly launch individual attacks.

D. BACKTRACK LINUX

We needed an operating system for both Snort and Pytbull. Snort can operate on either the Windows or Linux platform while Pytbull only operates on the Linux platform. We choose Linux as our operating system to simplify the software configuration so that we could install both applications on a single platform. We replicated the operating system and software configuration for multiple machines by using virtual machines.

BackTrack is a Linux-based intrusion-detection system/intrusion-prevention system penetration testing distribution that is free (BackTrack, n.d.). It provides security professionals with a large database of security tools packaged in the Linux operating system. We use BackTrack release 2 with KDE desktop environment. BackTrack can be installed and boot from a thumbdrive, harddrive, or directly from a Live DVD. A Live DVD refers to the ability to boot the entire operating system and run applications directly from a DVD.

BackTrack is pre-installed with 12 categories of security tools as shown in Table 3:

BackTrack intrusion-detection system/intrusion-prevention system penetration testing modules	
Information gathering	Stress testing
Vulnerability assessment	Forensics
Exploitation tools	Reporting tools
Privilege escalation	Services
Maintaining access	Miscellanous
Reverse engineering	
RFID tools	

Table 3. BackTrack Intrusion-detection System/Intrusion-prevention System Penetration Testing Modules

Both Snort and Pytbull, and their prerequisite tools (such as Tcpdump and Libpcap) are pre-installed in BackTrack. Therefore, we do not have to go through an entire package installation process. Software updating and rules updating (for Snort) is advised to ensure the latest package release is installed.

E. VIRTUALIZATION TECHNOLOGY

Virtualization software, such as the VMware, seeks to improve machine versatility by allowing a single machine to run multiple operating systems at the same time (VMware, 2012). A virtualization application runs on the main operating system, sharing the system resources with other applications. Multiple operating systems then run on the virtualization application. The resources allocated to the virtualization application are shared among these virtual machines. For example, the main operating system can be a running Microsoft Windows 7 operating system, while the virtual machines are running Linux operating systems. Virtual machines are installed on “virtual disk” residing on a separate file container on either the main machine or separate storage system. This separation ensures the files belonging to different virtual machines and main machine do not corrupt.

In our experiments, we used the VMware player version 5 as the virtualization software. This enabled us to use two physical machines to run six virtual machines at the same time. Snort and Pytbull were configured on these virtual machines.

F. EVALUATION CRITERIA

This section describes the metrics used to evaluate the prediction algorithms.

A true positive refers to making a correct positive prediction (the predicted event occurred) whereas a false positive refers to making a wrong positive prediction (the predicted event did not occur). A false negative refers to making a wrong negative prediction (the actual event coincides to the event that is predicted as not occurring). An intrusion detection alert prediction predicts the attacker IP address, the target IP address, the alert identification and the protocol type. These fields must match the real fields for the prediction to be considered a true positive prediction.

The precision measures the number of true positives in relation to the total number of positive predictions (sum of true positives and false positives) made (Rijsbergen, 1979). In cyber security, a high precision level is equivalent to predicting existence of real threats correctly with low levels of false alarms.

The recall measures the total number of true positive predictions in relation to the total number of actual positives (sum of true positives and false negatives). If the prediction in cyber security has a high recall value, we can say that the system focuses on security. That is to raise an alert for a possible threat than to miss a real threat.

The F-score is the harmonic mean of precision and recall. It rewards increases in both precision and recall. We use this metric for our prediction algorithm evaluation as it balances between precision and recall instead of sacrificing one metric for the other.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EXPERIMENTAL SETUP

In our alert dataset generation, we identified three key agents - the attacker, the victim, and the intrusion-detection system. The attacker uses Pytbull to launch various penetration tests on the victim, whereas the intrusion-detection system listens to the traffic and generates alerts in a log file. This chapter describes the hardware and software configuration used in our experiment.

A. EXPERIMENT SPECIFICATION

1. Hardware Specifications

The specifications of each hardware component in our experiments are listed in Table 4.

We deployed two physical machines in our networked environment. They were connected to the network switch via Ethernet cables. We use virtual machines on these computers to simulate multiple hosts on the network. The machine running the intrusion-detection system was connected to the mirrored port of the switch to listen to network traffic.

A broadband router acts as a gateway to the Internet. Its main function is to lease IP addresses to the virtual machines by acting as a Dynamic Host Configuration Protocol Server and to direct incoming and outgoing traffic through Network Address Translation.

Although our broadband router sufficiently connects the computers to form an internal network, it did not have a port mirroring feature to allow an intrusion-detection system to listen. So we deployed a mirroring capable Ethernet switch. The machine running intrusion-detection system was connected to port 1, and the other machine was connected to port 2. We configured all traffic from port 2 to be mirrored to port 1.

Machine 1 (Lenovo T500)	
Processor	Intel Core2 Duo P8600 2.4 GHz
Storage	240 GB
Memory	2 GB
Network Interface	Intel 82567LF Gigabit Network Connection
Operating System	Microsoft Windows XP Professional Service Pack 3
Machine 2 (Dell Latitude E6500)	
Processor	Intel Core2 Duo P8600 2.4 GHz
Storage	150 GB
Memory	4 GB
Network Interface	Intel 8256LM Gigabit Network Connection
Operating System	Microsoft Windows 7 Service Pack 1
Broadband Router (Cisco Linksys E4200)	
Standards	802.11a, 802.11b, 802.11n, 802.11g, 802.3, 802.3u, 802.3ab
Wireless Frequency Band	2.4 GHz, 5 Ghz
Network Ports	LAN: 4 x 10/100/1000 Mbps Ethernet Hi-Speed USB: 1 x 4 pin USB Type A WAN: 1 x 10/100/1000 Mbps Ethernet
Number of Antennas	6 antennas. 3 each per 2.4GHz and 5GHz radio band.
Ethernet Switch (Netgear ProSafe Plus 8-port Ethernet Switch GS108E)	
Standards	802.3i, 802.3u, 802.3z
Network Ports	LAN: 8 x 10/100/1000 Mbps Ethernet
Features	Network monitoring

Table 4. Hardware Specifications

2. Software Specifications

Initially, we configured Snort to run on the physical machine. However, we could not enable promiscuous mode for the network interface in Windows environment. Normally, a network interface only receives network packets designated to it; in promiscuous mode, the network interface accepts all network packets on the network. Thus, we ran Snort from within a Linux virtual machine,

which set the network interface in promiscuous mode during packet sniffing. We verified by checking that the Snort generated alert for attacks conducted on other virtual machines.

We installed VMware player 4.0.4 on both computers. We created three different types of virtual machines. Each virtual machine used 512 MB of memory and 14 GB of hard disk space. The virtual-network adapter was bridged to the physical network adapter. We then installed the same software for these virtual machines. We created multiple virtual machines by replicating the physical folder of the initial installation in other folders. A total of six virtual machines were deployed in our experiment.

The Linux-based penetration testing distribution, BackTrack 5 release 2, was installed as the operating system. The distribution uses KDE as the desktop environment and runs on a 32-bit CPU architecture. We opted for 32-bit instead of 64-bit because the physical machine used a 32-bit operating system. This also ensures portability across machines (or additional machines). To launch the desktop environment, we enter “startx” after the initial boot up sequence.

BackTrack was pre-installed with Snort and Pytbull. We updated Snort to version 2.9.2.3 and its prerequisite package Libpcap to version 1.0.0-6. There was a need to update Libpcap so that it is compatible to Snort. We also obtained the updated Snort ruleset release 2.9.2.3 from the Sourcefire Vulnerability Research Team. We enabled the ruleset in the Snort configuration file. We did not need to configure a SQL database for the Snort alert as we are using the default csv file logging.

We updated Pytbull to version 2.0. Prior to executing Pytbull or the Pytbull server, we must ensure Apache2, SSH and FTP services are already running as some of the attacks were conducted on these services. For both FTP tests and alert file retrieval, Pytbull requires the server to setup an FTP account and a user home directory. We also specify the paths of the supporting tools (nikto, hping3, ping, tcpreplay, ncrack, ab), which are necessary for Pytbull in the configuration

file. Pytbull launches specific tests according to the configuration file. The test configurations for each type of test were stored in module configuration file. Thus, we broke it down into individual tests by creating different module configuration files and Pytbull configuration files.

The test selection is achieved by executing the associated module configuration file and Pytbull configuration file. A script was created to select the desired test (or choose one at random) and to create continuous test runs. At the end of each Pytbull execution, it retrieved the Snort alert file via FTP and hosted a webpage to produce a summary of the intrusion-detection system/intrusion-prevention system penetration test. Since this feature was not required in our experiment, and to prevent the webpage hosting from halting our continuous test runs (running one test after another), we modified Pytbull codes to skip this feature.

Table 5 shows the three key members of our experiment (the attacker, the victim and the intrusion-detection system) and the software components they are using. They were virtual machines running within VMware Player 4.0.4.

Intrusion Detection System (+ Victim)	
Primary role	Sniff packets and generate alert log. Solicit attacks.
Software components	Snort 2.9.2.3 – intrusion-detection system Apache – web server SSH – secure shell server Vsftpd – ftp server Pytbull server – server to allow pytbull to conduct reverse shell commands
Victim	
Primary role	Solicit attacks.
Software components	Apache – web server SSH – secure shell server Vsftpd – ftp server Pytbull server – server to allow pytbull to conduct reverse shell commands
Attacker	
Primary role	Launch attacks to trigger intrusion-detection system alerts.
Software components	Pytbull – launch penetration test on victim machines

Table 5. Software Components on Experiment Machines

3. Network configuration

We deployed the network participants as virtual machines in our networked environment. A total of three attacker and three victim virtual machines were deployed, where one of the victims also had the intrusion-detection system running. We divided the virtual machines between the two physical machines to balance the load. Machine 1 hosted the intrusion-detection system, Victim 1 and Victim 2. Machine 2 hosted Attacker 1, Attacker 2 and Attacker 3. We determined that the intrusion-detection system required higher processing power as it processes all packets sniffed across the network. We allocated it to the victim virtual machines in one physical machine as victim machines are the receiving ends of the attacksand this did not require high processing power.

Machine 1’s network interface card connected to the mirroring port on the first port of the Ethernet switch. Machine 2’s network interface card connected to the second port. We connected the broadband router to port eight. All the virtual network interfaces (within the virtual machines) were bridged to the physical machine to simulate physical connections to the switch. We configured the switch to mirror all network packets from all other ports to port one. Figure 3 shows the network connections.

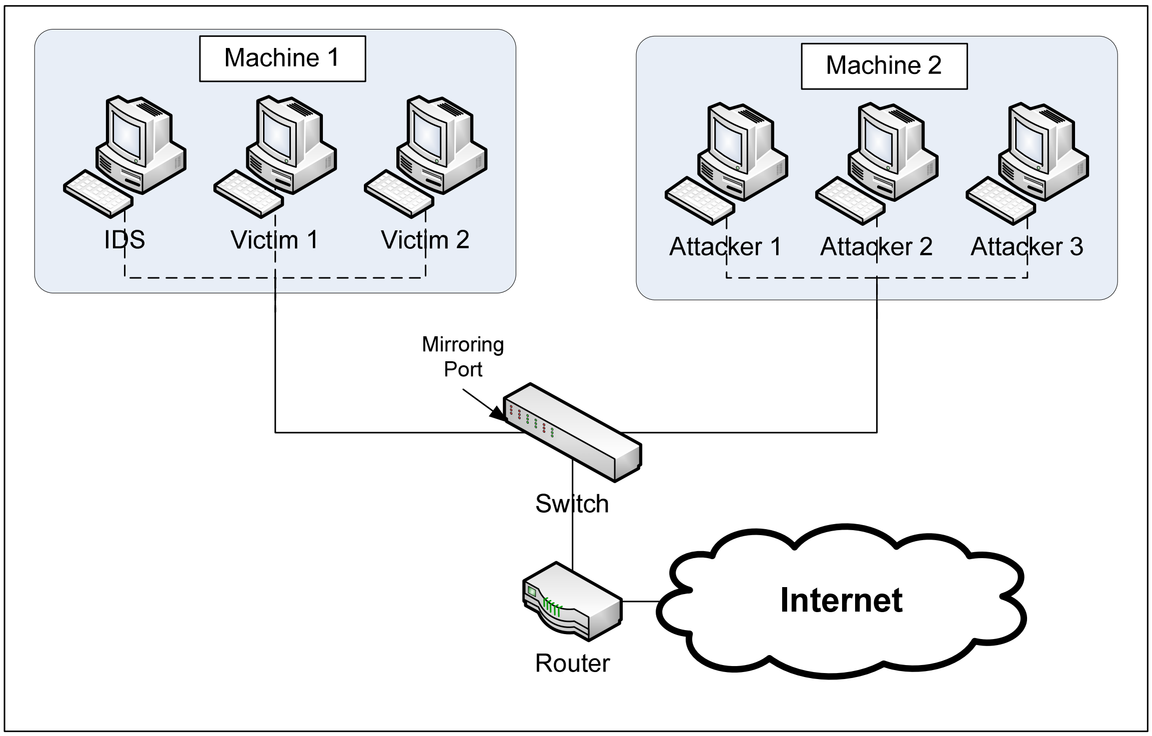

Figure 3. Network Connection Diagram

4. Problems Encountered

We initially deployed Snort in a Windows environment. The straightforward configuration is to put the Snort configuration directly on the physical machine. However, we realized that we need to determine whether it was working only after running the intrusion-detection system/intrusion-prevention system penetration testing tool.

Intrusion-detection system/intrusion-prevention system testing could be conducted by manually launching attacks. However, that is time-consuming and requires specific individual configuration such as port scanning followed by sending a payload. Pytbull presents itself as an automatic tester packaged with different types of test modules. Manual updating and configuration for Pytbull are required although it was preinstalled in the BackTrack distribution. For instance, we created a relevant user account on the operating system for the FTP service. The configuration file was also updated to reflect the folder path of the prerequisite tools Pytbull depends on. Pytbull launched tests in a fixed order. However, we wanted it randomized. We discovered that the tests are based on the module configuration file. We segregated these attacks into individual configuration files. At each Pytbull execution, we selected the configuration file to use by random.

After Pytbull was configured, we conduct some pilot runs. We discovered that Snort only detects network broadcast messages. Network traffic that was not directed to the Snort machine was not detected by Snort. Due to the limited resources, the participants were networked to the broadband router, which had a built-in switch. Online discussion sites revealed the possible reasons were either a lack of port mirroring switch, or the network interface's inability to operate in promiscuous mode. The following steps were taken to tackle the problem:

- Configured Linux-based Snort: We suspected that the physical machine's network interface could not operate in promiscuous mode in Windows environment. Since Snort is pre-installed in BackTrack, we reconfigured Snort to run from Linux environment.
- Snort Machine as victim: We conducted tests on the Snort machine to verify the configuration. We were unable to perform these tests on Windows-based Snort as the victim has to be running the Pytbull server in a Linux operating system.
- Connect the machines to a port-mirroring switch: We procured a port-mirroring-capable switch and configured mirroring in place of the broadband router's built-in switch.

- Tested Snort detection on victim: We started another virtual machine to test if Snort could detect Pytbull tests on other machine. We verified that this network configuration was working by checking the alerts for detected attacks on victim 1.

B. EXPERIMENT SETUP

In this section, we describe the different configurations we use to generate alert datasets.

1. Experiment One

We set up a scenario where there is one attacker targeting one victim. Between each attack execution, the attacker waits for a random period of up to 180 seconds. This is to randomize the frequency of attacks.

2. Experiment Two

We simulated a scenario where there are multiple attackers in the network. We increased the number of attackers to three and number of victims to three. The attackers launched their entire series of attacks one after another. But at any one time, there is only one attacker launching the attacks. Between each attack, the attacker waits for a random period of up to 180 seconds. Attackers randomly select the victims to attack. Because of this random selection, the number of attacker-to-victim pairs increases to nine pairs as compared to one pair in experiment one.

3. Experiment Three

In our third experiment, we evaluated the performance on the algorithm's prediction ability if the intrusion-detection system detects multiple attackers in randomized sequence. Three attackers were configured to launch attacks simultaneously. The wait period is a random period of up to 180 seconds.

4. Experiment Four

The last experiment evaluated whether the algorithm could perform prediction in situations where the attacks from different attackers overlap. Three attackers launch their attacks simultaneously with a wait period of about ten seconds.

5. Problems Encountered

We notice numerous “reset outside window” alerts were generated from Snort during our initial data collection. These alerts are repetitive and not meaningful. We are unable to explain the phenomenon other than by associating these alerts to the attacks that Snort is not able to detect and report. Snort is not able to detect client-side attacks and pcap replay (Albin, 2011). To reduce the number of “reset outside window” alerts, we disabled these Pytbull modules from launching during our experiment. Table 6 shows an example of a stream of “reset outside window” alerts Snort generats during client-side attacks.

Timestamp	Sig.	ID	Rev.	Message	Protocol	Source IP	Destination IP
06/19/12-21:02:55.614512	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.614516	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.614547	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.614551	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.615070	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.615078	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.620332	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101
06/19/12-21:02:55.620821	129	15	1	Reset outside window	TCP	192.168.1.137	192.168.1.101

Table 6. Reset Outside Window Alerts

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESULTS AND DISCUSSION

In this chapter, we present our results from running Tan's prediction algorithm programs, adapted from Tan and Darken (2012a, 2012b), on our generated dataset. We compared the prediction algorithms at different entropy levels.

Finally, we conducted significance testing to determine if the prediction accuracies of other algorithms were similar to that of the Single-Scope Blending algorithm.

A. ONE ATTACKER VERSUS ONE VICTIM

Figure 4 shows the accuracy of the prediction algorithm for one attacker and one victim. The dataset was divided into 126 batches of 100 percepts each. The F-score was used to evaluate the accuracy of the prediction algorithms. We post-processed and classified the prediction result of each batch based on different entropy levels. Entropy is a measure of uncertainty of random variables defined in Shannon (1984). In our context, the random variable is the occurrence of alerts. It is computed as

$$E = -\sum_{i}^{n} p(x_i) log_2 (p(x_i)),$$

where $p(x_i)$ is the probability of x_i. Entropy in each batch of percepts was used to represent the number of unique alerts (consisting of the attacker IP address, the target IP address, the alert identification and the protocol type). It describes the variability of that batch with regards to the proportion of each unique alert. The entropy increases with the number of unique alerts. Entropy is an appropriate measure because when entropy is low (highly repetitive and low number of unique alerts), many probabilistic and statistical prediction techniques would work well. Conversely, these techniques are expected to fail when the number of new alerts is large.

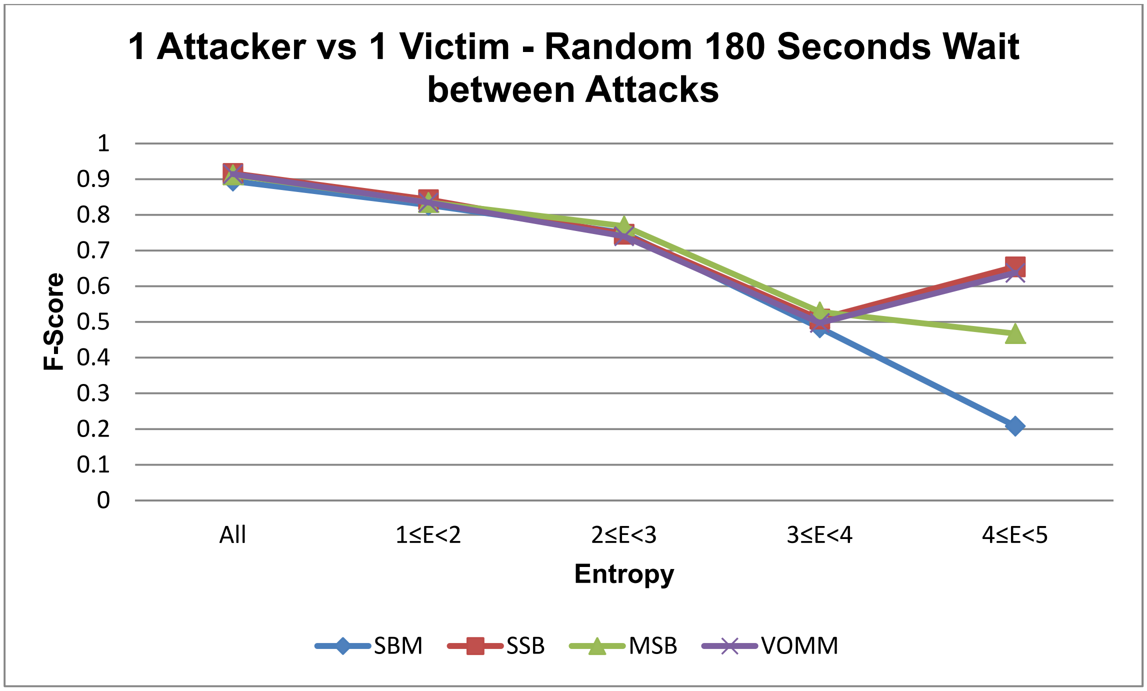

Figure 4. F-score Comparison: One Attacker versus One Victim with Random Period of up to 180 Seconds between Attacks

The accuracy of Single-Scope Blending (SSB) and Variable-Order Markov Models (VOMM) were similar across the entropy levels while Simple Bayesian Mixture and Multiple Simple Bayesian showed worse declining F-scores. We observed that F-score decreased as entropy increased. This was consistent with the unpredictability levels. However, there was a decrease in F-score at entropy level three for all algorithms, which had only two batches of percepts (Table 7) All four algorithms performed badly for one of the two batches, causing a sudden decrease in F-score .

	Entropy Level			
	1≤E<2	**2≤E<3**	**3≤E<4**	**4≤E<5**
Number of batches	13	18	2	6

Table 7. Breakdown of Percept Batches: One Attacker versus One Victim with Random Period of up to 180 Seconds between Attacks

B. THREE ATTACKERS VERSUS THREE VICTIMS

We wanted to compare prediction accuracy in scenarios where with more attackers and victims. We let three attackers randomly select one of three victims during each attack. This increased the possible actor pairs (attacker-victim) to nine to better simulate real-life cyber threats with multiple hackers scouring for potential victims on the network.

We observed no difference in the types of random attacks launched between a the first scenario and a three attackers versus three victims scenario. Figure 5 shows that SSB performed consistently better than the rest as we increased the number of attackers and victims in our experiment. At entropy level five, SSB is 0.1 better in F-score than MSB and VOMM. SBM, on the other hand, dd not show any change in accuracy.

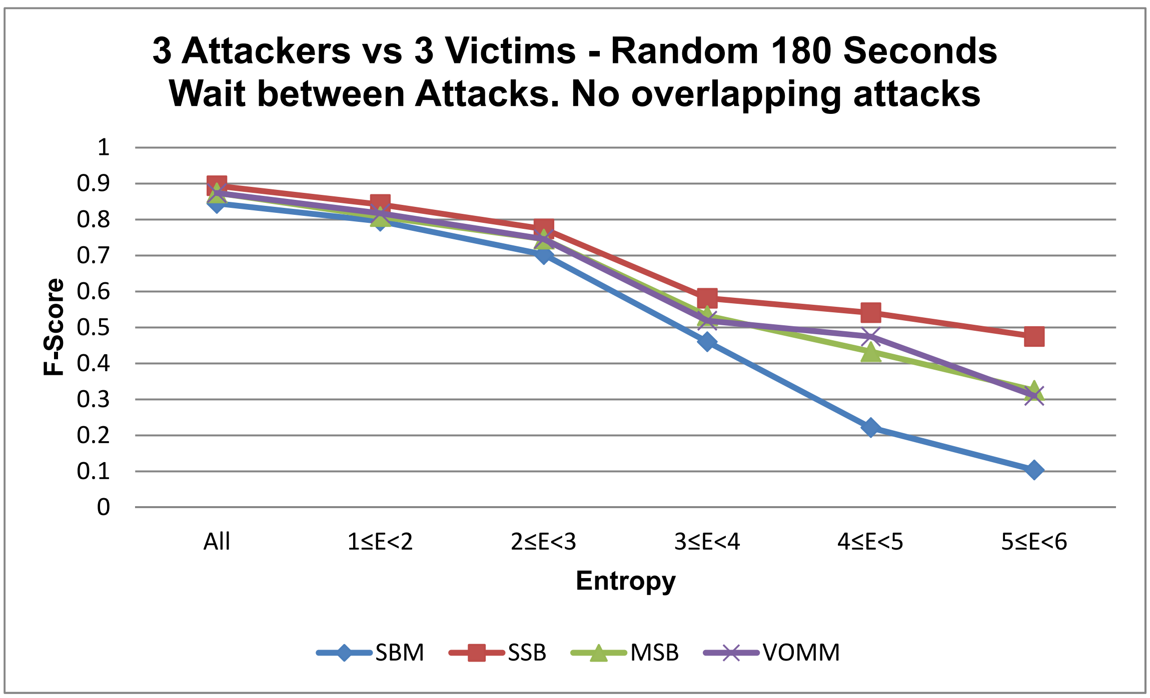

Figure 5. F-score Comparison: Three Attackers versus Three Victims with Random Period of up to 180 Seconds between Attacks (No Overlapping Attacks)

At lower entropy levels, we already observed different accuracies among the prediction algorithms unlike in the first scenario. The reason appears to be that SSB uses structural similarities to better match seemingly dissimilar situations. SBM, MSB and VOMM received incoming percepts as nine different actor pairs. Assuming there was already a set of situations in the knowledge base involving attacker-victim pair A-B performing action set X, as SSB received incoming percepts involving a different attacker-victim pair C-D but performing same action set X, it could cast an analogy from A-B to C-D to make a prediction (Tan & Darken, 2012b). SBM, MSB and VOMM cannot form this analogy, because they can only predict alerts that have been generated before.

Figure 6 shows the results where we allowed three attackers to launch their attacks consecutively. We expected the prediction complexity to increase because of the increased probability of overlapping attacks from different attackers, but a 180 seconds wait between attacks was too large to show a new effect.

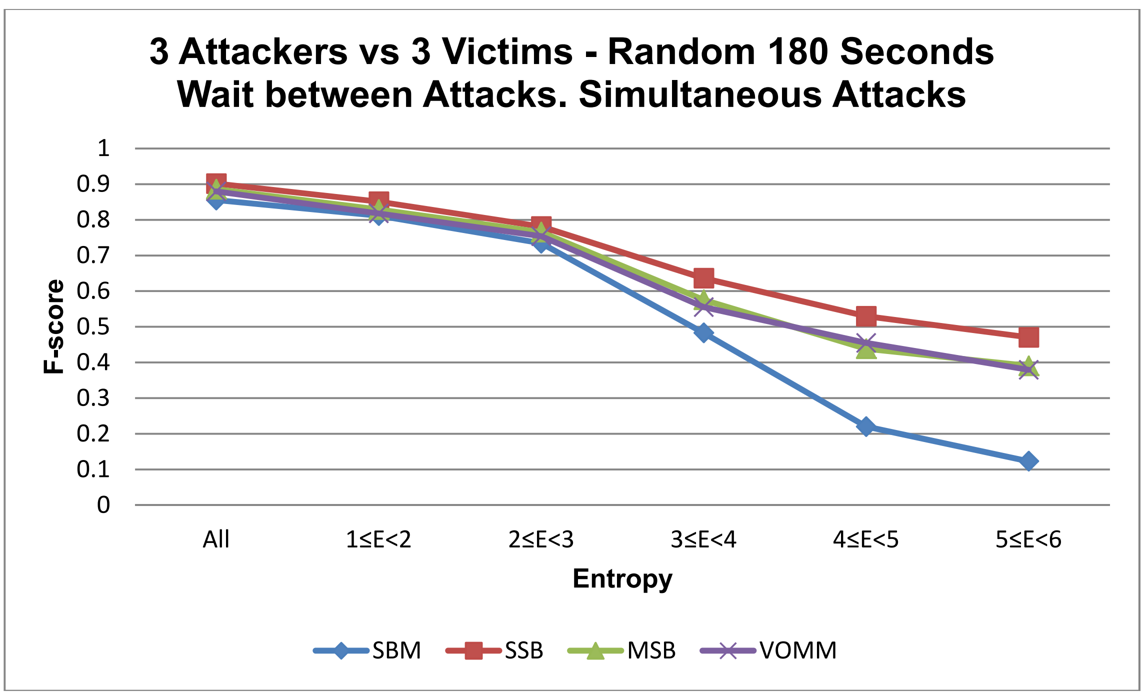

Figure 6. F-score Comparison: Three Attackers versus Three Victims with Random Period of up to 180 Seconds between Attacks (Simultaneous attacks)

Table 8 shows the number of alert batches distributed over the different entropy level. We reduced the random wait time between attacks to about 10 seconds. At the same time, the number of incoming percept sequences increased within a short time period. Table 9 shows an extract of alerts from attackers of different IP addresses. It was observed that other attackers launched attacks while IP address “192.168.1.115” sent fragmented packets to IP address “192.168.1.101. Other than SBM whose accuracy remained similar to previous attack configurations, all other prediction algorithms showed declined accuracy at entropy level five. SSB remained the top performer by at least 0.08 as shown in Figure 7.

		Entropy Level				
		1≤E<2	**2≤E<3**	**3≤E<4**	**4≤E<5**	**5≤E<6**
Number of batches	**Experiment 2**	32	30	33	26	3
	Experiment 3	75	101	57	46	11
	Experiment 4	100	140	75	65	15

Table 8. Distribution of Percept Batches for Experiment 2,3 and 4

Timestamp	Sig.	ID	Rev.	Message	Protocol	Source IP	Destination IP
07/27/12-08:55:01.345651	123	13	1	(spp_frag3) Tiny fragment	TCP	192.168.1.115	192.168.1.101
07/27/12-08:55:01.345844	123	13	1	(spp_frag3) Tiny fragment	TCP	192.168.1.115	192.168.1.101
07/27/12-08:55:03.695396	1	17322	1	SHELLCODE x86 OS agnostic fnstenv geteip dword xor decoder	TCP	192.168.1.138	192.168.1.102
07/27/12-08:55:03.695396	1	1378	21	FTP wu-ftp bad file completion attempt	TCP	192.168.1.138	192.168.1.102
07/27/12-08:55:03.695396	125	2	1	(ftp_telnet) Invalid FTP Command	TCP	192.168.1.138	192.168.1.102
07/27/12-08:55:03.766376	1	2000001	0	FTP brute force failed login unicode attempt	TCP	192.168.1.117	192.168.1.101
07/27/12-08:55:04.009447	1	1122	10	WEB-MISC /etc/passwd	TCP	192.168.1.117	192.168.1.101
07/27/12-08:55:04.323224	123	13	1	(spp_frag3) Tiny fragment	TCP	192.168.1.115	192.168.1.101
07/27/12-08:55:04.323327	123	13	1	(spp_frag3) Tiny fragment	TCP	192.168.1.115	192.168.1.101

Table 9. Extract of Alerts from Multiple Attackers

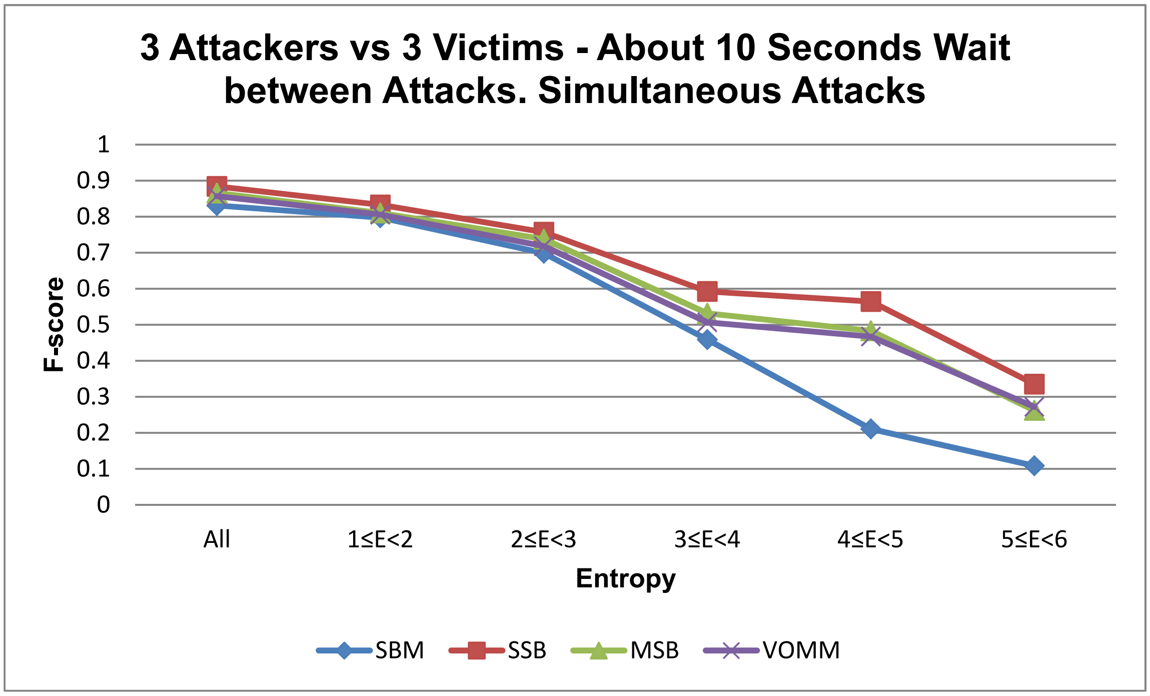

Figure 7. F-score Comparison: Three Attackers versus Three Victims with Random Period of about 10 Seconds between Attacks. (Simultaneous Attacks)

Although SSB outperforms the other prediction algorithms, it was the slowest. The computation data is tabulated in Table 10. The maximum prediction time for SSB was 4.283 seconds as compared to the MSB (next best performer) at 0.141 seconds, while the mean prediction time was 0.221 seconds for SSB compared to 0.011 seconds for MSB.

	Time (seconds)			
	SBM	**SSB**	**MSB**	**VOMM**
Maximum	1.182	4.283	0.141	0.065
Mean	0.038913	0.220718	0.010814	0.017035

Table 10. Computation Time: Three Attackers versus Three Victims with Random Period of about 10 Seconds between Attacks. (Simultaneous Attacks)

Tables 11, 12, and 13 provides the probability the prediction accuracies of other algorithms were similar to that of SSB. Paired T-Test compares F-scores of each situation while group t-test compares the average F-Scores. These results

support the hypothesis (at 95% confidence level) that SSB outperforms SBM, MSB, and VOMM from a different perspective.

	1≤E<2		
	SBM	**MSB**	**VOMM**
Paired T-test	5.77E-04	1.36E-02	6.54E-19
Group T-test	5.68E-03	3.51E-02	7.20E-20
	2≤E<3		
	SBM	**MSB**	**VOMM**
Paired T-test	1.25E-08	6.00E-04	2.49E-05
Group T-test	4.12E-05	3.02E-02	3.05E-02
	3≤E<4		
	SBM	**MSB**	**VOMM**
Paired T-test	5.35E-10	1.80E-03	1.42E-07
Group T-test	4.12E-08	1.71E-02	1.55E-03
	4≤E<5		
	SBM	**MSB**	**VOMM**
Paired T-test	7.33E-11	4.97E-07	4.68E-05
Group T-test	1.40E-15	1.45E-03	2.14E-02
	5≤E<6		
	SBM	**MSB**	**VOMM**
Paired T-test	6.45E-06	1.34E-02	2.65E-03
Group T-test	1.30E-07	3.23E-02	1.19E-02

Table 11. T-test Probabilities that the Algorithm's Performance is similar to that of SSB's (Experiment 2)

	1≤E<2		
	SBM	**MSB**	**VOMM**
Paired T-test	8.65E-15	3.55E-09	9.53E-15
Group T-test	1.59E-06	1.75E-03	1.31E-05
	2≤E<3		
	SBM	**MSB**	**VOMM**
Paired T-test	1.71E-12	8.07E-05	2.98E-09
Group T-test	1.23E-05	3.63E-02	1.48E-03
	3≤E<4		
	SBM	**MSB**	**VOMM**
Paired T-test	1.49E-29	6.06E-15	2.80E-21
Group T-test	1.37E-19	1.35E-05	1.17E-08
	4≤E<5		
	SBM	**MSB**	**VOMM**
Paired T-test	2.39E-15	2.56E-13	1.73E-10
Group T-test	1.18E-22	4.47E-04	6.64E-03
	5≤E<6		
	SBM	**MSB**	**VOMM**
Paired T-test	3.18E-07	6.49E-06	7.88E-06
Group T-test	3.30E-11	1.15E-02	9.95E-03

Table 12. T-test Probabilities that the Algorithm's Performance is similar to that of SSB's (Experiment 3)

	1≤E<2		
	SBM	**MSB**	**VOMM**
Paired T-test	8.34E-24	1.34E-14	1.20E-14
Group T-test	5.17E-04	1.77E-02	2.92E-03
	2≤E<3		
	SBM	**MSB**	**VOMM**
Paired T-test	8.71E-20	9.12E-06	1.37E-14
Group T-test	1.05E-08	1.50E-02	1.46E-05
	3≤E<4		
	SBM	**MSB**	**VOMM**
Paired T-test	5.56E-28	5.56E-28	6.77E-11
Group T-test	1.75E-17	1.75E-17	9.24E-05
	4≤E<5		
	SBM	**MSB**	**VOMM**
Paired T-test	4.02E-18	4.02E-18	5.89E-16
Group T-test	3.58E-25	3.58E-25	9.91E-04
	5≤E<6		
	SBM	**MSB**	**VOMM**
Paired T-test	5.22E-07	5.22E-07	1.74E-04
Group T-test	3.91E-08	3.91E-08	4.26E-02

Table 13. T-test Probabilities that the Algorithm's Performance is similar to that of SSB's (Experiment 4)

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION AND FUTURE WORK

This thesis generated intrusion-detection system alerts to support the performance analysis of prediction algorithms in cyber security. We generated intrusion alerts by simulating attacks within an internal network environment. This provided a sufficient dataset for the evaluation of a prediction algorithm, although some Pytbull modules are not detected by Snort. This approach saves effort because it would be time-consuming to collect intrusion alerts from real attacks through honeypots. We were able to adjust the frequency of attacks, number of attackers and number of targets to help us in our evaluation.

We then evaluated the performance of several relational time-series prediction algorithms on our generated alerts. The prediction accuracy declined as the entropy level of the alerts increased. We observed that an increase in number of attackers and victims lowered accuracy of prediction, except of SBM, which underperformed consistently. The performance of MSB and VOMM were similar across the experiments, and inferior to that of Single-Scope Blending. It appeared that the latter's conceptual blending approach was able to make good use of the structural properties during situation selection. It could help with situations where attackers vary IP address and targets.

For future work, we suggest implementing online prediction algorithms into an intrusion-detection system. We could set up a common database to allow the system to store new alerts while prediction algorithms retrieve and process them. The predicted alerts can be stored in a prediction database to provide network administrators with additional information.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Albin, E. (2011). *A comparative analysis of the Snort and Suricata intrusion-detection systems.* (Master's thesis, Naval Postgraduate School). Retrieved from http://edocs.nps.edu/npspubs/scholarly/theses/2011/September/11Sep_Albin.pdf

BackTrack. (n.d). *BackTrack*. Retrieved from BackTrack Linux – Penetration Testing Distribution website: http://www.backtrack-linux.org

Cipriano, C., Zand, A., Houmansadr, A., Kruegel, C., & Vigna, G. (2011). Nexat: A history-based approach to predict attacker actions. *ACSAC '11 Processings of the 27th Annual Computer Security Applications Conference,* 383–392.

Council, N. S. (n.d.). *Cybersecurity*. Retrieved from the White House website: http://www.whitehouse.gov/cybersecurity

Damaye, S. (2012). *Pytbull*. Retrieved from Pytbull website: http://pytbull.sourceforge.net/index.php?page=home

Darken, C. J. (2005). Towards learned anticipation in complex stochastic environments. *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference,* 27–32.

Geib, C. W., & Goldman, R. P. (2001). Plan recognition in intrusion detection systems. *Proceedings of the Second Darpa Information Survivability Conference and Exposition (DISCEXII),* 329–342.

Johnson, N. B. (2011, March 23). *Attacks on Federal networks increased forty percent.* Retrieved from Federal Times website: http://www.federaltimes.com/article/20110323/IT01/103230303/

Lei, J., & Li, Z. (2007). Using network attack graph to predict the future attacks. *Communications and Networking in China, 2007*, 403–407.

Li, Z., Zhang, A., Li, D., & Wang, L. (2007). Discovering novel multistage attack strategies. *ADMA '07 Proceedings of the 3rd international conference on Advanced Data Mining and Applications*, 45–56.

Olney, M. (2008). *Performance rules creation.* Retrieved from Snort website: http://www.snort.org/assets/173/SnortUsersWebcast-Rules_pt1.pdf

Rijsbergen, C. V. (1979). *Information retrieval*. Retrieved from http://www.dcs.gla.ac.uk/Keith/Preface.html

Rowe, N. C., Custy, E. J., Duong, B. T. (2007). Defending cyberspace with fake honeypots. *Journal of Computers, 2,* 25–36.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal, 27,* 379–423, 623-656.

Snort. (2012). Snort. Retrieved from www.snort.org

Spitzner, L. (1999). *Build a honeypot*. Retrieved from http://www.spitzner.net/honeypot.html

Symantec. (2011, April 5). *Symantec report finds cyber threats skyrocket in volume and sophistication*. Retrieved from http://www.symantec.com/about/news/release/article.jsp?prid=20110404_03

Tan, K. M., & Darken, C. J. (2012a). Learning & prediction in relational time series: A survey. *21st Behavior Representation in Modeling & Simulation (BRIMS) Conference 2012*, 93–100.

Tan, K. M., & Darken, C. J. (2012b). Faster conceptual blending predictors on relational time series. *Information Fusion (FUSION), 2012 15th International Conference*, 188–195.

VMware. (2012). *Virtualize your IT infrastructure*. Retrieved from http://www.vmware.com/virtualization/

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Professor Yeo Tat Soon
   Director, Temasek Defence Systems Institute
   National University of Singapore, Singapore

4. Ms Tan Lai Poh
   Senior Manager, Temasek Defence Systems Institute
   National University of Singapore, Singapore

5. Mr Teo Tiat Leng
   Deputy Director, Land Systems
   Defence Science and Technology Agency, Singapore

6. Mr Khong Farn Wei Jason
   Naval Postgraduate School
   Monterey, California